Evaluating Early High-Performance Computing Systems

Tarek El-Ghazawi, P.I.
Armagan Ozkaya and Abdullah Meajil, GRAs

Dept. of Electrical Engineering and Computer Science
The George Washington University
Washington, D.C. 20052

FINAL REPORT: June-December 1993
Submitted to USRA/CESDIS

INTRODUCTION

Evaluating high-performance computing systems (HPCS) based on MPP architectures is a key part of the HPCC initiative. In order to provide HPCS designers with useful insight, one must correlate performance measurements with the dynamic characteristics of the workload. To do so, a model for experimentally characterizing workloads should be developed. This model should be based on a set of attributes that capture how the workloads stress the various architectural features.

In this work, an abstract model has been developed where the characterization attributes are designed to reflect applications scalability, spatial and temporal locality of references for both instructions and data, and amount of work. The experimental characterization has been conducted in a manner that isolates the effects of the machine loading or architecture on which the measurements are obtained focusing only on the application.

This report includes an overview of our abstract

characterization model, the tools that have been ported and used, ESS problems subjected to our experiments, tools developed, and the results of the experiments conducted.

ESS WORKLOAD CHARACTERIZATION MODEL

In preparation for the first release of the ESS Parallel Benchmark (EPB) set, a detailed characterization of the ESS workload have been performed with special emphasis on quantitative evaluation of resource requirements and scaling properties. ESS workload characteristics would be very useful in assembling EPB from key code kernels, applications, and synthetic load generators.

Desirable workload model characteristics include:

* orthogonal descriptors
* architecture-independent characteristics which are functions of the application only and not of the architecture or problem mapping.
* architecture-significant attributes which can well describe the required machine resources as well as the application behaviors that machine designers can capitalize on.
* observable
* informative and easy to understand
* accurate

Our workload model, which is given below, has evolved in accord with the previous criteria. This model can be seen as an n-tuple of functions of address space "s", time "t", and input size"i", as follows:

W=<Pc,Pd,Lit,Lis,Ldt,Lds,P-avg,Idy>

where,

Pc(t) : Control parallelism (profile or smoothability)
Pd(t) : Data parallelism (profile or smoothability)
Lit(t) : Temporal locality of instruction references
Lis(s) : Spatial locality of instruction references
Ldt(t) : Temporal locality of data references
Lds(s) : Spatial locality of data references
P-avg(i) : Average instruction parallelism
Idy(i) : Dynamic instruction count

TOOLS PORTED/USED

In order to capitalize on related efforts by other researchers, a number of performance analysis and visualization packages had been investigated, located, ported, analyzed, and used to produce measurements.

Below are brief explanations of some of the interesting packages:

ParaGraph

ParaGraph is a tool to visualize the behavior and performance of parallel programs using the message-passing paradigm. It has been developed by the University of Illinois. It takes as input a trace data in a specific format, and uses it to provide performance statistics in a graphical format. These trace data are generated during an actual run of a parallel program on a message-passing machine, which can then be replayed pictorially with ParaGraph to display a dynamic, graphical depiction of the behavior of the program.

ParaGraph presents performance statistics on: (1) utilization of processors, (2) communication overhead, (3) performance of concurrent tasks. In each subarea (util, commun, tasks), ParaGraph allows the user to choose exactly in what form he/she would like to view the statistics; such options include a whole variety of graphs ranging from Gantt charts to Kiviat diagrams to Streak graphs etc. Of 25 different graphs (views), most can depict up to 512 processors, although a few are limited to 128 processors and one is limited to 16. ParaGraph is based on the X Window System, runs on a wide variety of graphical workstations, and allows responsive interaction with the user.

SPA

SPA (stands for SPARC Performance Analyzer) is a set of tools, developed by Gordon Irlam of Sun Corp., to analyze the performance of SPARC binaries on the SPARCstation-1 and SPARCstation-2. The tools can be used on any Sun4 architecture machine running a SunOS 4 (preferably 4.1) operating system.

The SPA package does not predict overall system performance. I/O latency is not taken into consideration, nor is the effect of more than one process being active at any one time. Although SPA has some other deficiencies as well, it includes tools that are shown to be convenient to build around:

Spy that traces the execution of a command. It can be used to generate an address trace which can then be passed directly to a trace analyzer, and/or list the system calls performed by that command. For each system call, the attributes of the process being traced, the address of the system call, the name of the routine being called, and some other useful information are extracted.

Traced processes have been observed to run about 40 times slower than normal. If Spy is used with a trace analyzer, the resulting system will run some 600 times slower than normal.

Spanner that reads the address trace input, analyzes it to see how often each instruction occurs, and writes out this information, along with information on simulated cache misses and other events. It provides options to specify the type of processor, cache and write-back buffer, and other architectural attributes.

SITA

SITA (SPARC Instruction Trace Analyzer) is another set of tools used to analyze SPARC executables. Its purpose is to determine the amount of parallelism that may be extracted by an idealized machine model which is configured with certain user-definable capabilities. It was developed by McGill University and is designed to work with Spy tool. SITA includes a pre-analyzer, a control-dependence analyzer, and a trace scheduler.

The analysis of a SPARC program takes four steps. First, a SPARC program is created. The results will be more meaningful if the program is statically linked. Second, the pre-analyzer is run to extract a list of basic blocks of the program which is then read by the control dependency analyzer to produce a list indicating the control dependency relationships between the blocks, as the third step. Finally, the scheduler is run with this list as input. The scheduler produces output showing the parallelism available for the given input trace under the given model(s). It is only necessary to perform first three steps a single time for any given program. Once the output of the control dependency analyzer becomes available, it may be used for all subsequent parallelism experiments for various machine models.

SITA tools are mentioned below in the order they are used to extract parallelism figures.

Sitapa (SITA pre-analyzer) is an auxiliary tool providing a total of four functions that can be desired. Its primary function is to analyze a dynamic trace to determine the boundaries of the basic blocks (a basic block is a sequence of instructions that is always entered at the beginning and exited at the end), and to record the frequencies of branch outcomes. This function will also record some useful statistics about the

trace: the total number of executed/annulled instructions; the number of memory accesses; the number of floating-point operations; deepest depth reached at the stack; the total number of basic blocks in the program, both static (in the program) and dynamic (in the trace).

Second function of Sitapa is to act as a disassembler which outputs a mnemonic ASCII form of the dynamic trace. Inversely, Sitapa can convert traces in this mnemonic ASCII form into a certain form in order to be applied to the other tools in SITA. The final option is to combine one or more lists of basic blocks produced by earlier runs of Sitapa on the traced program into a single block list. This ability is required mainly because of the fact that a single run of Sitapa on the traced program may not yield a complete list of possible basic blocks.

Sitadep (SITA dependency analyzer) performs control-dependency analysis on a list of basic blocks which is generated by Sitapa. It produces an annotated list of basic blocks and control-dependency information which consist of, among others, the total number of procedures, the number of instructions in the longest basic block, the most common destinations from the blocks, and the critical blocks (those on which other blocks are control dependent).

Sitarun is the trace scheduler. This tool reads in a dynamic trace and schedules the instructions according to an idealized parallel machine(s) based on user-definable parameters. The total (instruction level) parallelism is measured and reported along with the description of the model and some statistics on prediction rates and speculation depth. The parallelism is computed by dividing the total number of useful sequential operations (the work) by the number of parallel instructions needed to schedule these operations (the time), which are both given in the output.

It is possible to run multiple experiments (for different machine models) simultaneously using the same trace. Configuration parameters may specify such attributes of the model as the type of branch prediction, the speculation depth, the maximum number of operations per parallel instruction, latencies of various classes of instructions, and the use of memory renaming/disambiguation.

ESS PROBLEMS and CODE WRITTEN

Programs of sequential code for selected ESS problems have been written in order to get measurements of attributes given in our characterization model:

Backpropagation

A tool which can "learn" different transfer functions for solving different problems is an invaluable tool. Neural networks have this ability of "learning" which makes them ideal to use in a wide range of problems, particularly in classification and recognition related ones.

Our implementation is a simulation of a feedforward backpropagation neural network with two hidden layers. The number of neurodes in input/output layers are problem-dependent, but there is no general rule to determine hidden layer(s) sizes. Neurodes in the network have nonlinear transfer functions whose nature play an important role in the performance of the network.

The basic idea is that given an input and output pair, a learning rule is used to adjust the connection weights so that the network is more likely to give that output when it receives that input. This "training" process involves calculation of neurode outputs based on neurode inputs and weights, backward

error propagation, and weight adjustment calculations. These calculations can be done in two different fashions; the network could be "trained" on a pattern-by-pattern basis or over an entire set of patterns (called "epoch"). This sequence of operations continues for a number of times (iterations) until the network converges, that is, the error factor becomes less than a projected value. After training, the response of the network for a particular input set can be determined by calculating neurode outputs based on the final weights and this input set.

The N-body Problem

The study of physical systems by particle simulation is called the "many-body" or the "N-body" problem. N-body simulations have been used to study a wide variety of astrophysical systems during the past years, ranging from small clusters of stars to galaxies and the formation of large-scale structure in the universe. Such studies are conducted in celestial mechanics, plasma physics, and fluid mechanics, as well as in semiconductor device simulation.

The simulation finds the trajectories of each particle over some time interval, given the initial position, the initial velocity, the external forces, and the nature of the forces that the particles exert on each other. Thus, the simulation proceeds over a large number of time-steps, in every time-step computing the net force on every particle and updating its position and other attributes.

Because the N-body problem can not be solved in closed form, the calculation must be done numerically. That is, after each time interval, the gravitational forces of each mass on each of the others may be computed by Newton's laws. Newton's law of gravity states that the force between any pair of particles is proportional to the product of their masses divided by the square of the distance between them. When there are many particles, the acceleration of each particle is given by the sum of accelerations induced by all the other particles. By many iterations of this method of calculation, the position of each particle after an arbitrary length of time may be found.

There are different algorithms to solve this problem. In the simplest method, where each particle interacts with each other particle, all pairwise forces are computed directly which involves the calculation of all n(n-1)/2 forces. The time complexity is O(n**2) where n is the number of particles.

Therefore, this method allows an accurate description of the dynamical evolution but at a price that grows rapidly for increasing number of particles.

The Barnes-Hut method uses a tree-structured hierarchical subdivision of space into cells, each of which is recursively divided into eight subcells whenever more than one particle is found to occupy the same cell. Although the calculations can be performed more quickly -the complexity is O(n.log n)-, the result will be of less accuracy.

In the Miller method, the solution technique represents the problem in a position-velocity space, and transforms the force field using a fast Fourier transform into a form where it can be applied in linear time. This method is of complexity O(n.log n) as well.

The Aaerseth method involves in keeping track, for each particle, of the sets of "nearby" particles and "faraway" particles. The "faraway" particles may be integrated with large time-steps than the "nearby" particles. When the particles are uniformly distributed, the time complexity is O(n**1.5), otherwise O(n**2).

In the Hockney method, the universe is divided into cells. First the particle-to-particle interactions are computed, and then the cell-to-cell interactions are handled. This technique has complexity O(n**4/3) for a uniform distribution. There is a variation of Hockney method, of complexity O(n.log n), where the cell-to-cell interactions are computed using the fast Fourier transform.

Morphological Filtering

Infrared cirrus emission is diffuse extended structure in the Galaxy which prevents the exploration of the extragalactic sky at far-infrared wavelengths. A grayscale morphological filter is now being used by members of the NASA HPCC/ESS science team for the removal of such cirrus emission. Due to the computational requirements of the problem, high performance computing testbeds are used for the implementations. Image morphology is based on two fundamental operations, dilations and erosions, that are described by the following algorithm. These operations use template called the structuring-element to operate on the target image as described below.

The serial algorithm for dilation is now given. An image (I) and a structuring element (se) are of sizes NxNxG and MxMxG, respectively, where graylevels are integers ranging from 0 to G-1.

```
Algorithm DILATE-SER:
Begin
for i=1 upto N
        for j=1 upto N
           position center of se "se(0,0)" on the top of I(i,j)
           I_tmp(i,j) = - 1
           for k=1 upto M
              for l=1 upto M
                if(I(i,j)+se(k,l))>I_temp(i,j))
                   I_temp(i,j) = I(i,j) + se(k,l);
               endfor;
            endfor;
        endfor;
endfor;
End.
```

The algorithm visits each pixel in the NxN image, and uses the

graylevel at this pixel to modify all pixel levels in the MxM structuring elements. Clearly, an O(N^2*M^2) algorithm.

The erosion algorithm is similar, except that the ">" becomes "<" and, the "+" becomes "-". Therefore, the serial erosion algorithm is also an O(N^2M^2) as well. Since the morphological filter consists of sequences of dilations and erosions, then the filter has also an O(N^2M^2) time complexity. For this same reason, only the dilation algorithm is considered for performance discussions.

TOOLS DEVELOPED

SPAL

We have developed a package of five tools, which we call SPAL (SPA-Locality) to analyze the instruction/data spatial/ temporal locality characteristics of a program and the relative locations of read/write systems calls in the program. These tools, for which the brief descriptions are given below, use the outputs of SPA and SITA tool sets.

Spasl-i (SPA Spatial Locality-Instruction) extracts instruction spatial locality in terms of the number and percentage of dynamic instructions located in each address space slice. To find out locality of instruction references in a program, Sitapa must be run first whose output including basic blocks and the number of visits to each block will be the input to Spasl-i. Also, the number of address space slices into which the whole instruction address space will be divided must be specified.

Spasl-d (SPA Spatial Locality-Data) extracts spatial locality of program data in terms of number and percentage of data accesses to an address space slice. To find out the locality of data references, the executable code must be run with Spy and Sitapa, which will give the disassembled version of the dynamic trace, then the output should be piped to Spasl-d. The number of address space slices into which the whole data address space will be divided must be specified.

Spatl-i (SPA Temporal Locality-Instruction) finds out the temporal locality of program instructions. The program must be traced by Spy, these traces must be disassembled by Sitapa whose output in turn will be applied to Spatl-i. Some additional attributes such as the number of dynamic instructions, the number of time windows, and the threshold

value must be specified by the user.

Spatl-d (SPA Temporal Locality-Data) finds out the temporal locality of program data. It has the same specifications and usage as Spatl-i.

Spa-io (SPA Input/Output) uses as input the disassembled version of the program (produced by Sitapa) and the list of the system calls executed (produced by Spy) during the program run. Spa-io performs three basic functions: (1) Determines the number of occurrences and the relative locations of traps in the program; (2) Determines the read/write system calls, their associated files, and the amount of data read from/written into these files; (3) Determines the relative locations (in terms of time) of read/write system calls in the program. The last two functions allow us to see the execution time instances as well as intensities of input/output activities in the program.

CONCLUSIONS

Our results include the degree and variability of instruction level parallelism, spatial and temporal locality of instructions and data, and some input/output characteristics of ESS applications.

Diagrams showing the results regarding the parallelism, smoothability (variability), and locality characteristics of the selected applications are given in the Experimental Results section of Appendix-1. Our conclusions from these diagrams are:

* Temporal locality is, generally, higher for data.
* Locality properties are invariant to problem size.
* Backpropagation seems to have better spatial locality. Moreover, the independence in operating on the training sets resulted in close to linear increase in parallelism. The

critical path length in Backpropagation, however, seems not to grow as rapidly with increase in the input, which resulted in decreasing smoothability.

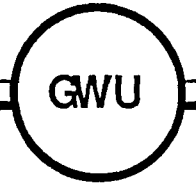
* For N-body, it seems that in the used algorithm, the critical path length grows more rapidly than parallelism with the increase in dynamic instructions count. This implies poor scalability for larger number of bodies. As the input size increases in N-body, parallel instructions increase and start to dominate the execution producing less increase in variability.

It should also be noted that for Backpropagation both network size and the number of training vectors should, intuitively, have effect on the parallelism, which is what we have observed. However, because of the limitations in the implementation of the SITA, such as the maximum speculation depth, after a certain network and/or input size, parallelism figures decrease with these factors. Our Backpropagation program is composed of a sequence of matrix manipulation procedures most of which are data dependent to previous ones. As the problem size increases, loops in these procedures become longer separating parallel operations by large distances, which makes it difficult for SITA to discover them. Consequently, parallel instructions can not be utilized efficiently.

Appendix-1

The poster paper published in Supercomputing'93 Conference.

This poster consists of: brief information about the abstract workload characterization model and two selected applications - Backpropagation and N-body problem; tools used for experiments; diagrams showing the results for parallelism, smoothability, spatial and temporal locality properties of these applications; and the conclusions.



NASA SCIENCE WORKLOAD CHARACTERIZATION FOR SCALABLE PARALLEL COMPUTER ARCHITECTURES

Thomas Sterling*, Tarek El-Ghazawi**, Armagan Ozkaya**, and Abdullah Meajil**

*Center of Excellence in Space Data and Information Sciences
Code 930.5
NASA Goddard Space Flight Center
Greenbelt, MD 20771

**Department of Electrical Engineering and
Computer Science
The George Washington University
Washington, D.C. 20052

Supercomputing'93 , Portland, Oregon, November 15-19

Abstract

CESDIS

GWU

Background and Motivation:

- The NAS Parallel Benchmarks (NPB) have had a significant impact on the assessment of high performance computer architectures for aerospace engineering.
- A new initiative sponsored by NASA is intended to produce for the Earth and space sciences (ESS) a benchmark that offers what NPB has provided for NASA's engineering disciplines.
- ESS domain is distinguished by computational challenges that stress memory systems and I/O systems dynamics.

Approach:

- In preparation for the first release of the ESS Parallel Benchmark set (EPB) a detailed characterization of the ESS workload is being performed with special emphasis on quantitative evaluation of resource requirements and scaling properties.
- ESS workload characteristics should be used in assembling EPB from key code kernels, applications, and synthetic load generator(s).
- ESS workload characteristics should be also used in interpreting the benchmark results

Abstract (Cont.)

CESDIS

GWU

Tools Used:

- SITA, developed at McGill University, uses a trace of the application under study and performs extensive data dependency analysis.
- Trace data acquired includes dynamic instruction profiling, procedure and system calls profiling, and memory address tracing. The analysis provides instruction mixes, higher-level memory access profile, and program dependency information.

Results:

- Degree and variability of instruction-level parallelism
- Spatial and temporal locality of data and instructions
- Results also used in the Joint NSF/NASA Initiative on Evaluations

1. Workload Model

Desirable Workload Model Characteristics:

- Orthogonal descriptors
- Architecture-independent-- measurements are function of the application only and not the architecture or the problem mapping.
- Architecture-significant-- attributes can well describe the required machine resources as well as the application behaviors that machine designers can capitalize on.
- Observable
- Informative
- Accurate
- Easy to understand

Workload Model (Cont.)

CESDIS

GWU

Our initial workload model, which is given below, will evolve in accord with the previous criteria. This workload characterization model "W" is currently seen as an n-tuple of functions of address space "s", time "t", and input size "i", as follows:

W = <Pc, Pd, Lit, Lis, Ldt, Lds, P-avg, Idy>

where

Pc(t)	-- Control parallelism (profile or smoothability)
Pd(t)	-- Data parallelism (profile or smoothability)
Lit(t)	-- Temporal locality of instruction references*
Lis(s)	-- Spatial locality of instruction references*
Ldt(t)	-- Temporal locality of data references*
Lds(s)	-- Spatial locality of data references*
P-avg(i)	-- Average instruction parallelism as a function of the input size*
Idy(i)	-- Dynamic instruction count as a function of the input size*

Measurements only for those attributes that are followed with asterisks are reported here.

2. Two Example ESS Applications

N-Body

Definition:

Study of physical systems by particle simulation is called the "many-body" or the "Nbody" problem.

Applications:

Study a wide variety of astrophysical systems (from small clusters of stars to galaxies and formation of large-scale structures in the universe).

Strategy:

Given the initial position, the initial velocity, the external forces, and the nature of the forces that the particles exert on each other:

- Proceed over a large number of time-steps
- In every step compute the net force on every particle, update its position and other attributes
- Find trajectory of each particle

Two Example ESS Applications (Cont.)

Backpropagation

- An implementation of feedforward back-propagation neural network with two hidden layers.
- Neurodes have nonlinear transfer functions (in our program tanh) whose nature play an important role in the performance of the network.
- Given an input/output pair, a learning rule is used to adjust the (connection) weights so that the network is more likely to give that output when it receives that input.
- The "training" process involves calculation of neurode outputs based on neurode inputs and weights, backward error propagation, and weight adjustment calculations.
- We use batch, or "epoch", training method where the error calculation is done on a neurode-by-neurode basis over the entire set (epoch) of presentations(patterns), rather than on a pattern-by-pattern basis.

3. Measurement Technique and SITA

CESDIS

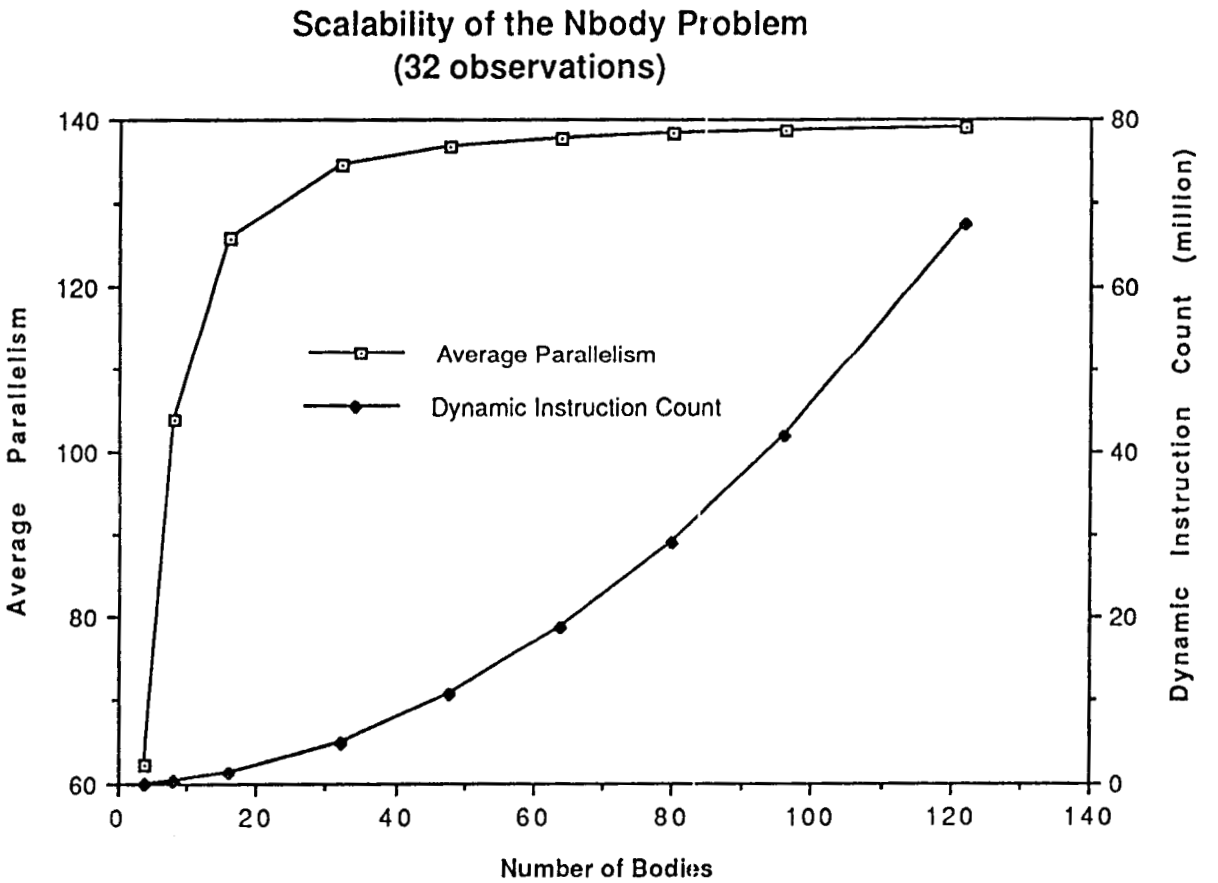
GWU

- Measurements were obtained from SPARC implementations of the studied applications, with the aid of SITA (developed by the Advanced Compilers, Architectures, and Parallel Systems group at McGill University).
- SITA is a set of tools used to analyze SPARC executables to determine the amount of parallelism that may be extracted by an idealized machine model.
- SITA is designed to work with SPY tool, which is a part of the SPA(SPARC Performance Analyzer) package.
- SITA first extracts a list of basic blocks and frequencies. Control-dependency relationships and destination frequency are then figured out. At this point the scheduler is run, producing an output indicating the available parallelism.
- SITA dependency analysis output and spy address trace files were used by modules developed locally (spasl and spatl) to study the locality of references for the underlying applications.

4. Experimental Results: Nbody

CESDIS

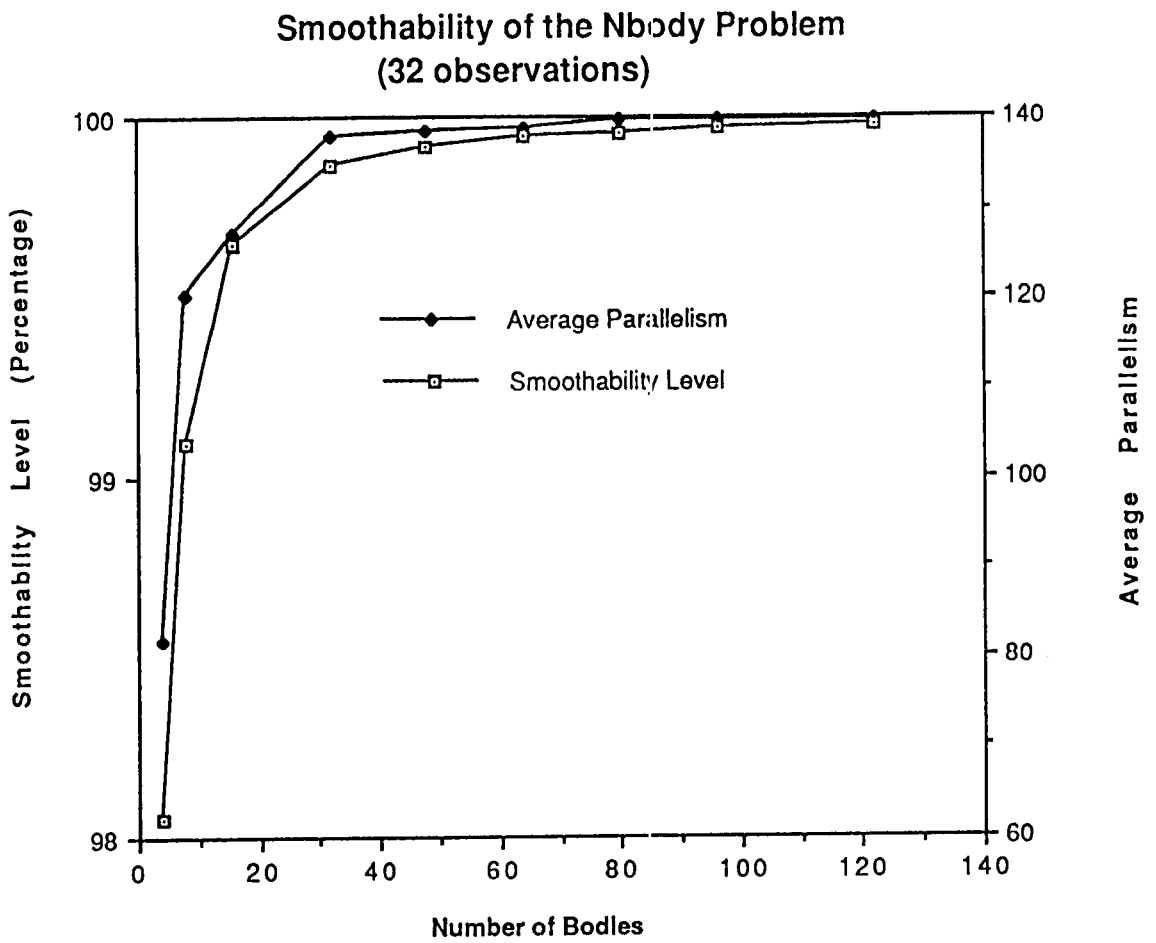
GWU



4. Experimental Results: Nbody (Cont.)

CESDIS

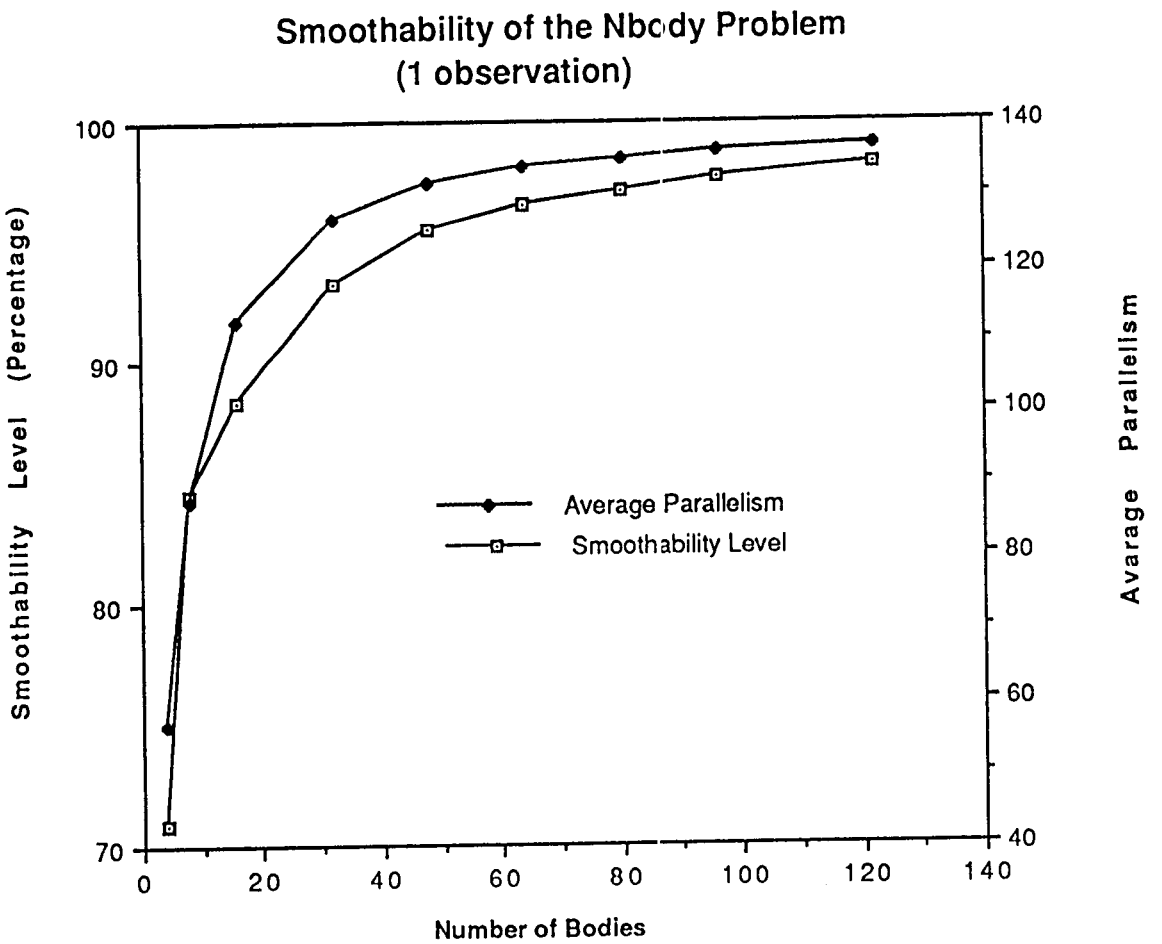
GWU



4. Experimental Results: N-body (Cont.)

CESDIS

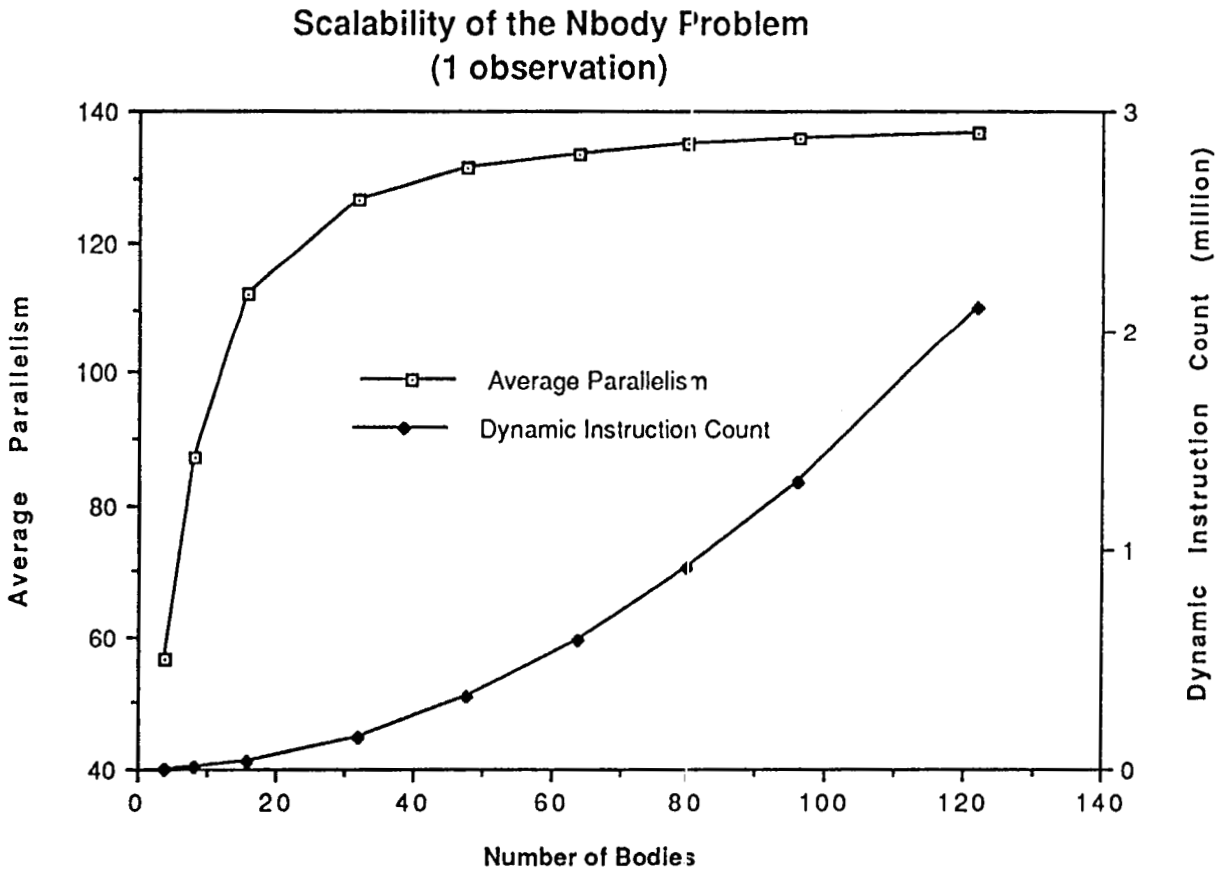
GWU



4. Experimental Results: Nbody (Cont.)

CESDIS

GWU

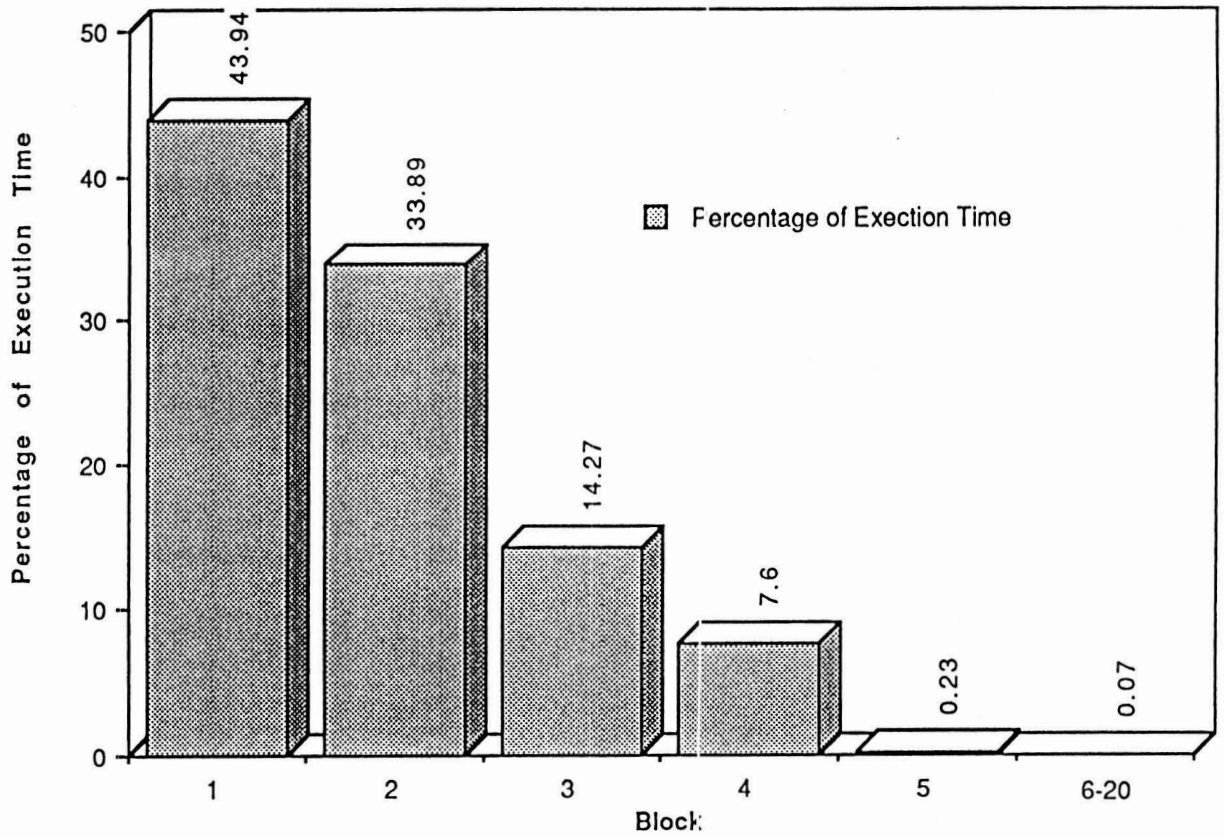


4. Experimental Results: Nbody (Cont.)

CESDIS

GWU

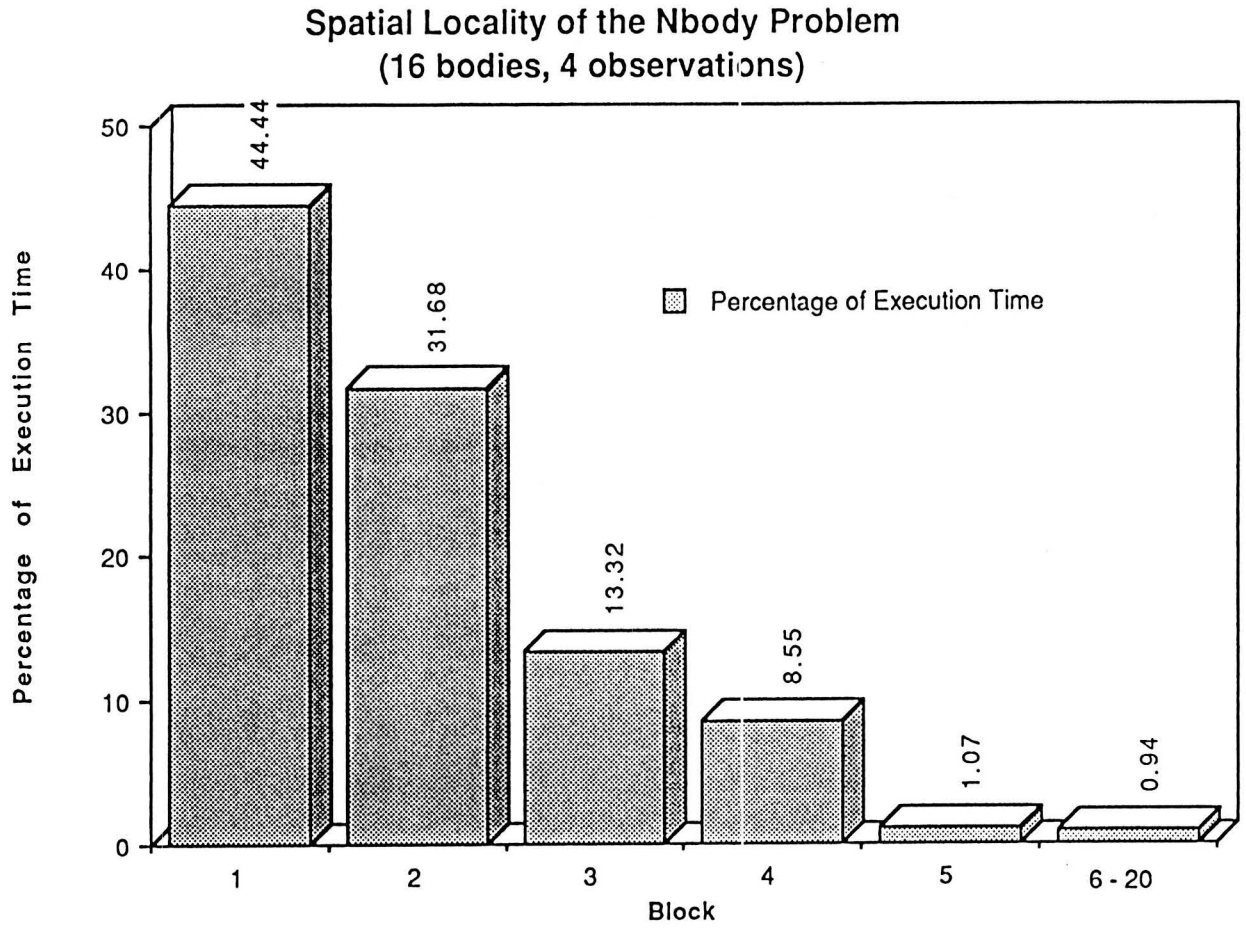
Spatial Locality of the Nbody Problem
(64 bodies, 4 observations)



4. Experimental Results: N-body (Cont.)

CESDIS

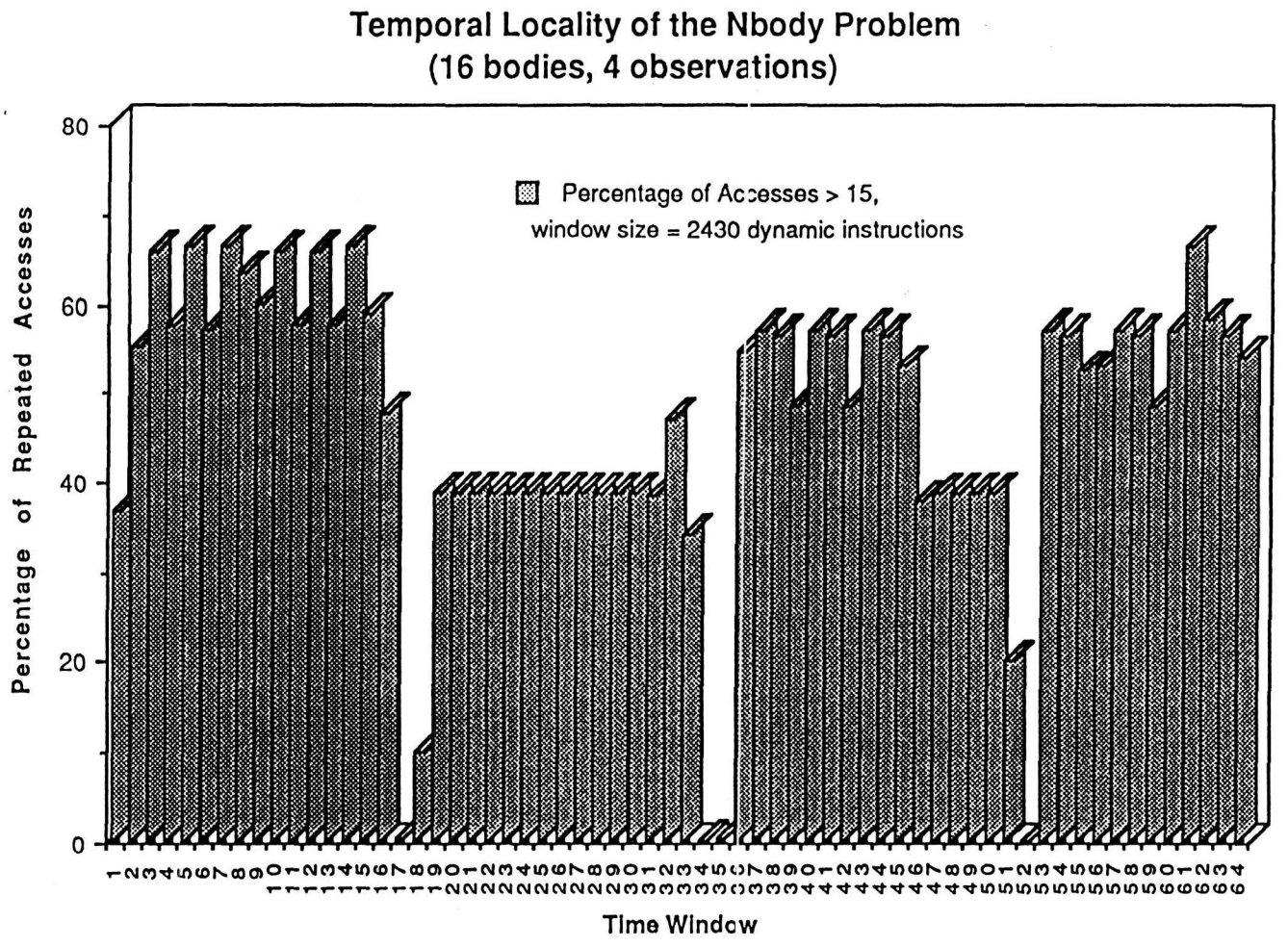
GWU



4. Experimental Results: N-body (Cont.)

CESDIS

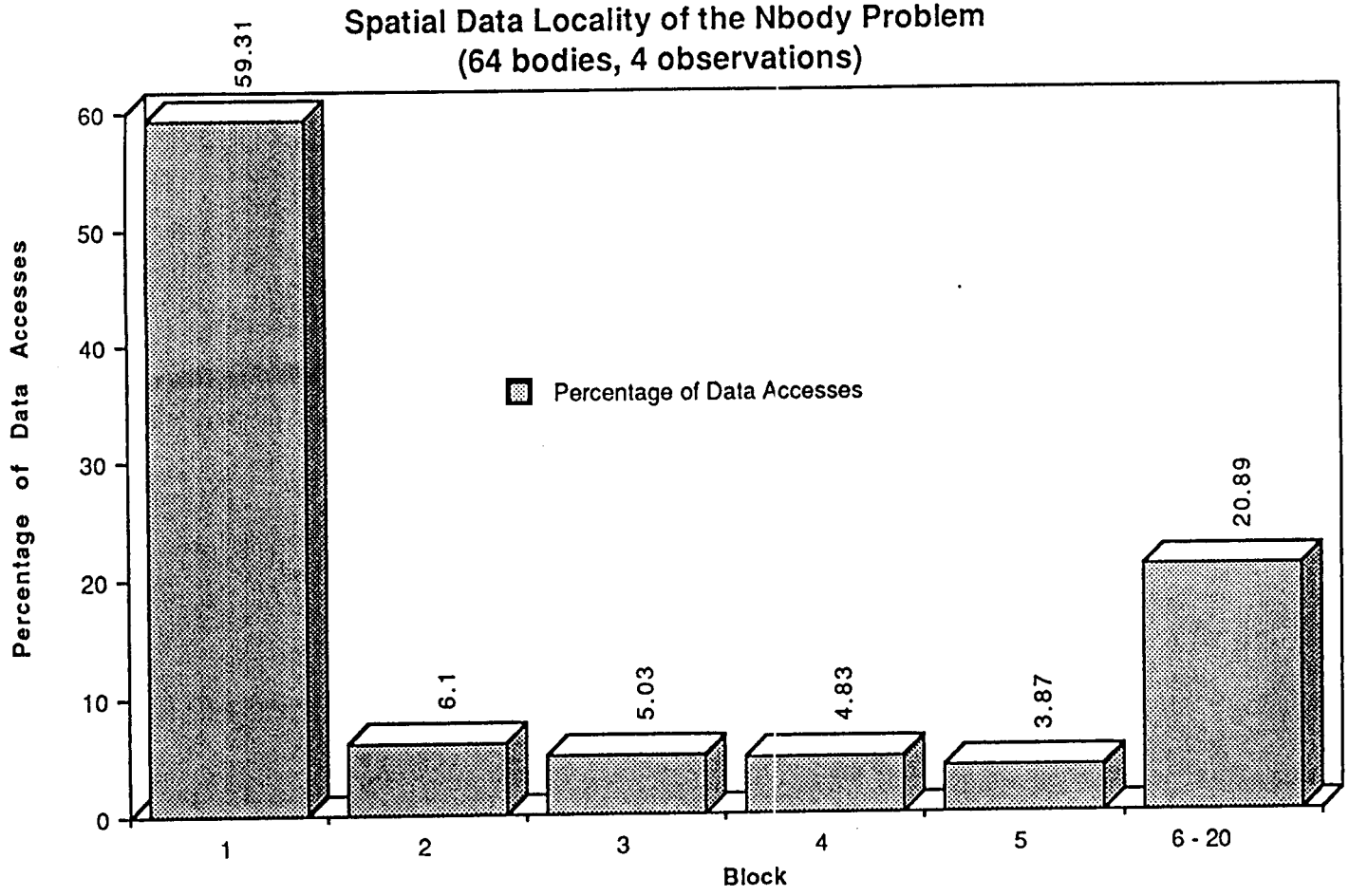
GWU



4. Experimental Results: N-body (Cont.)

CESDIS

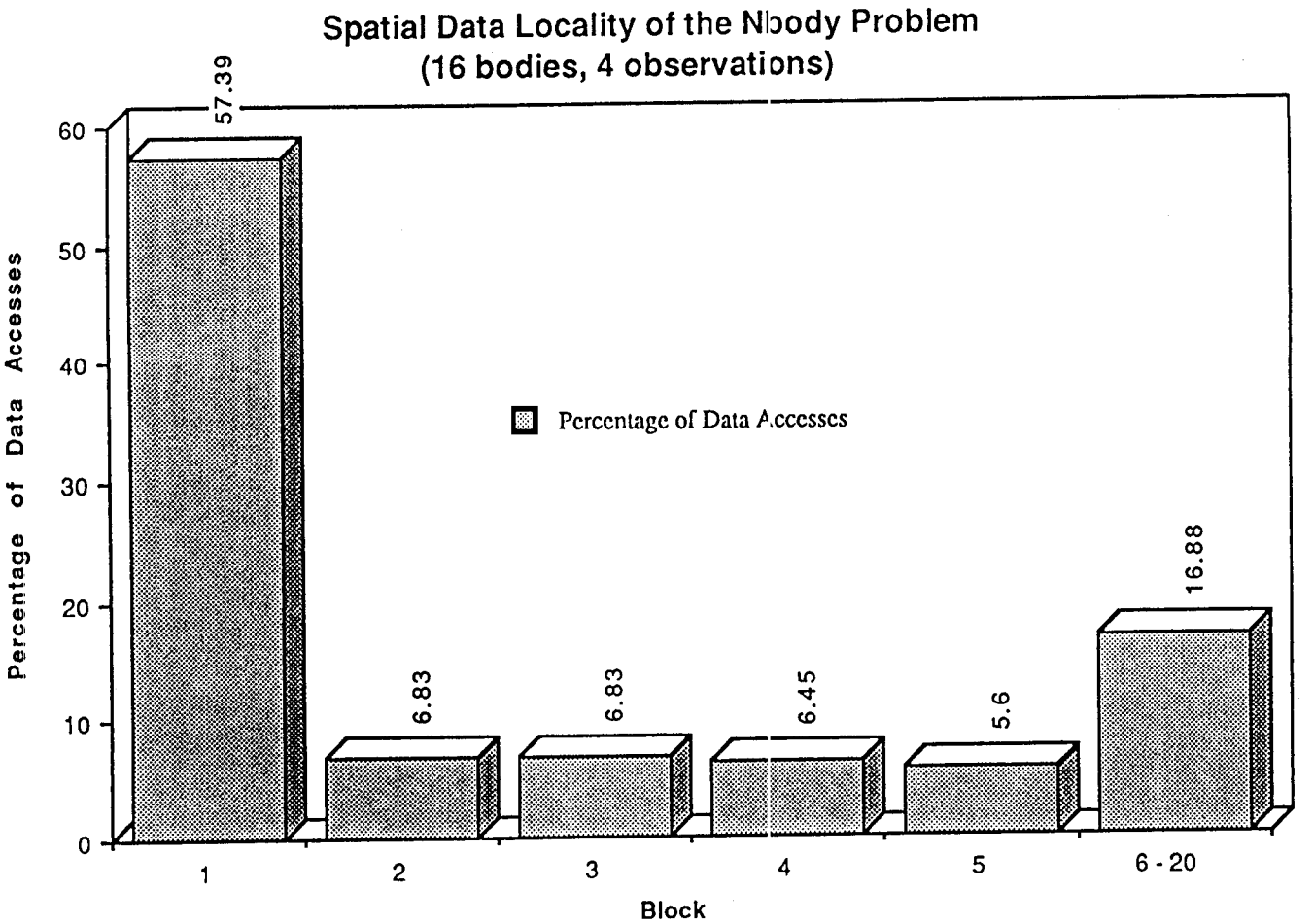
GWU



4. Experimental Results: N-body (Cont.)

CESDIS

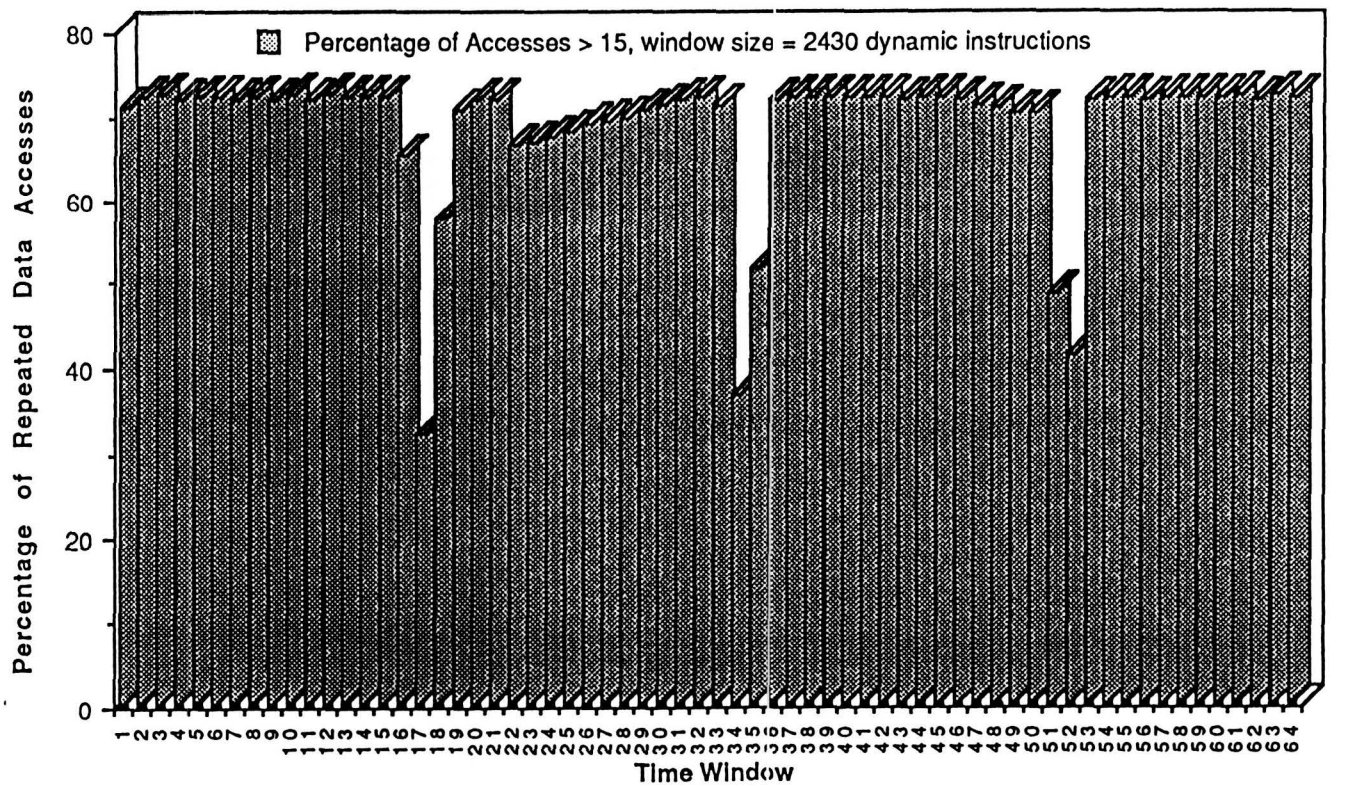
GWU



4. Experimental Results: N-body (Cont.)

CESDIS

GWU

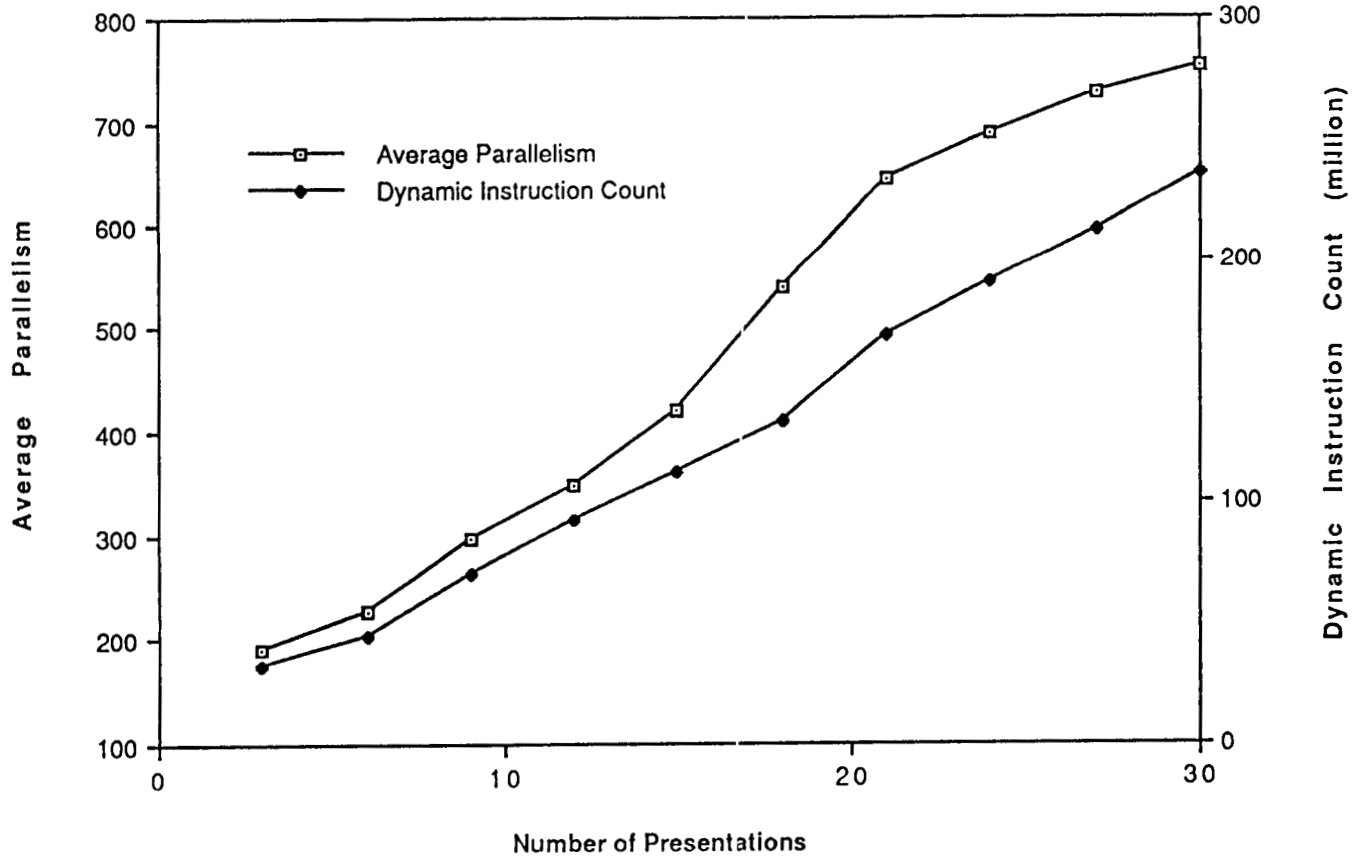


4. Experimental Results: Backpropagation

CESDIS

GWU

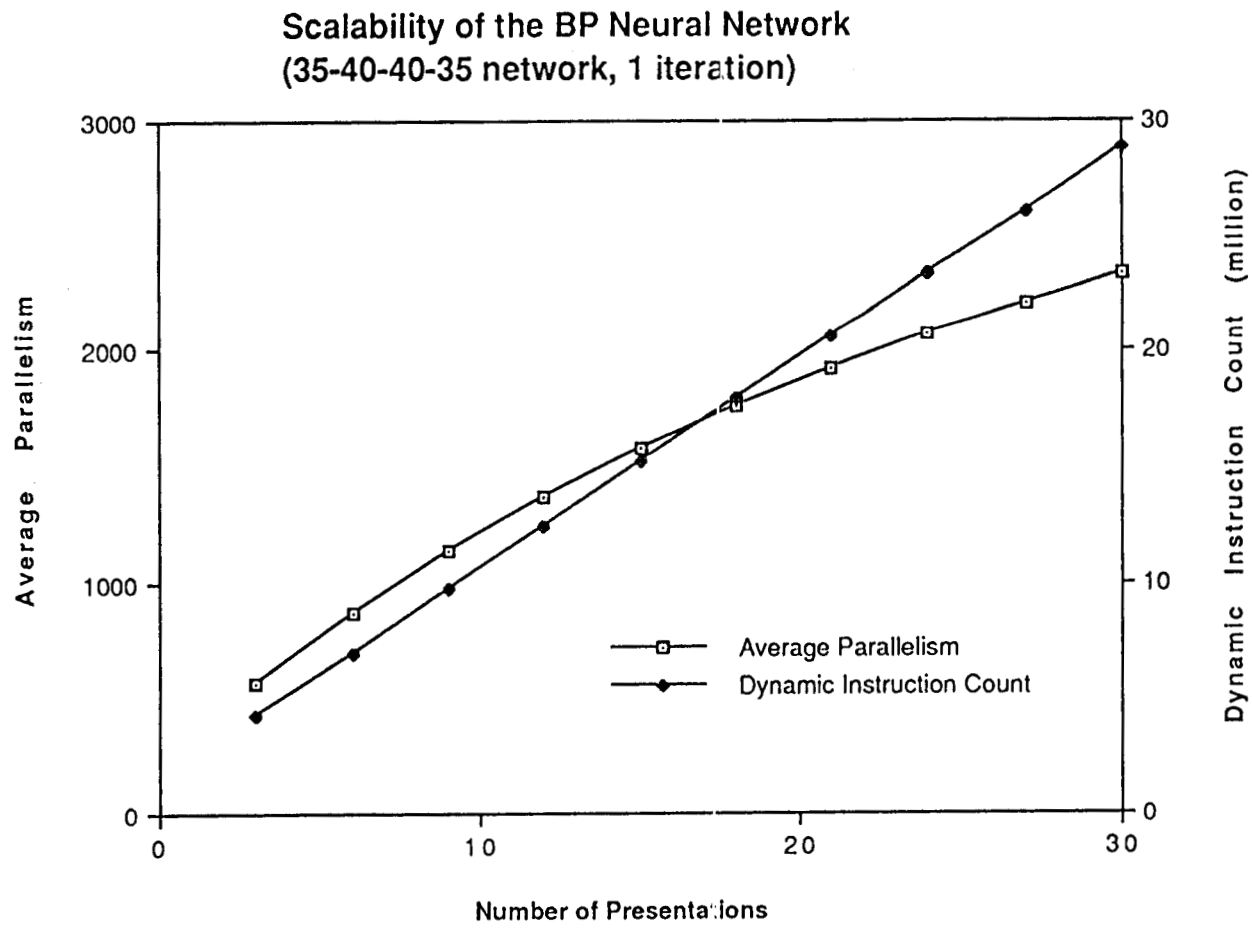
Scalability of the BP Neural Network
(2-3-3-3 network, 1000 iterations)



4. Experimental Results: Backpropagation (Cont.)

CESDIS

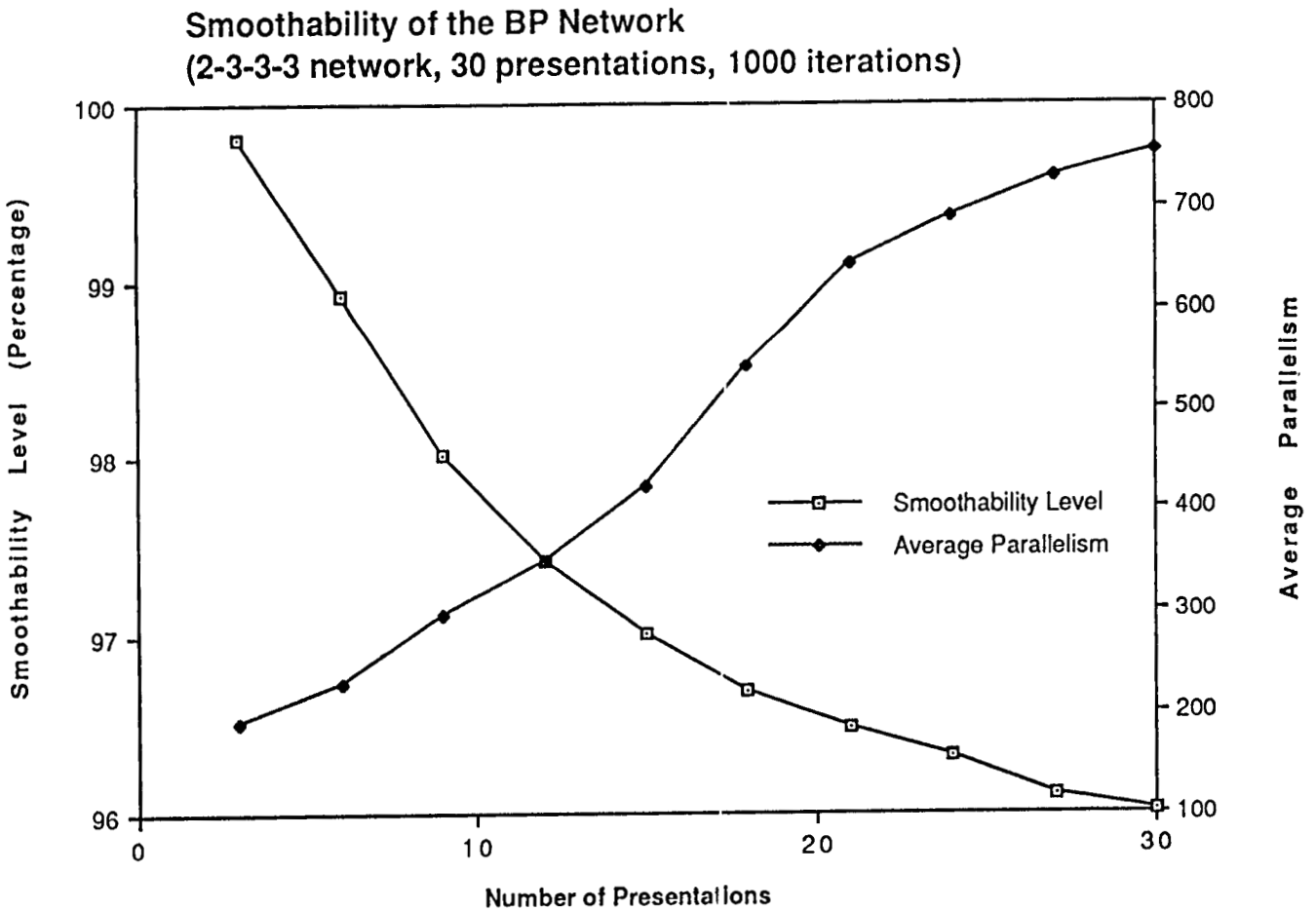
GWU



4. Experimental Results: Backpropagation

CESDIS

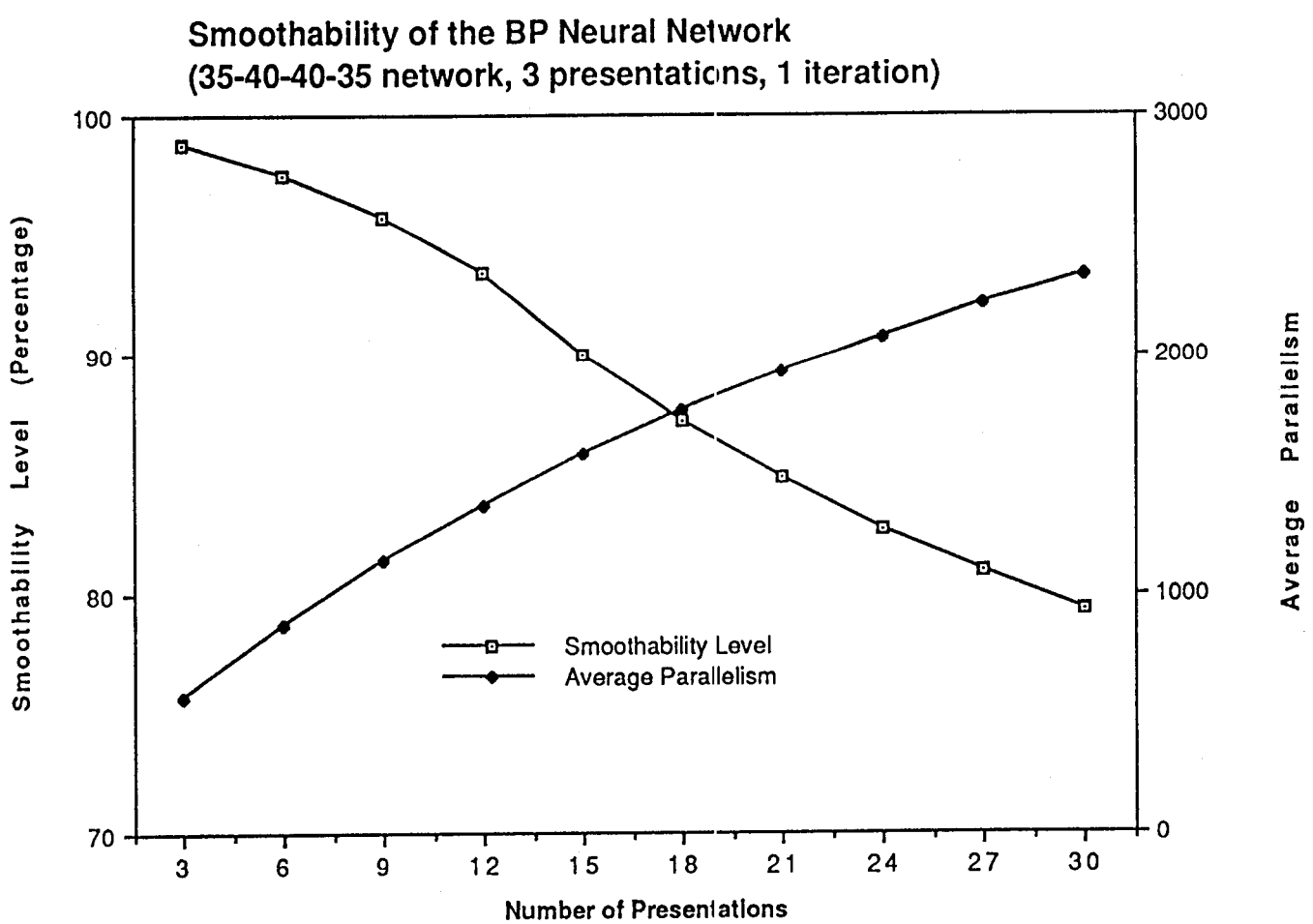
GWU



4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

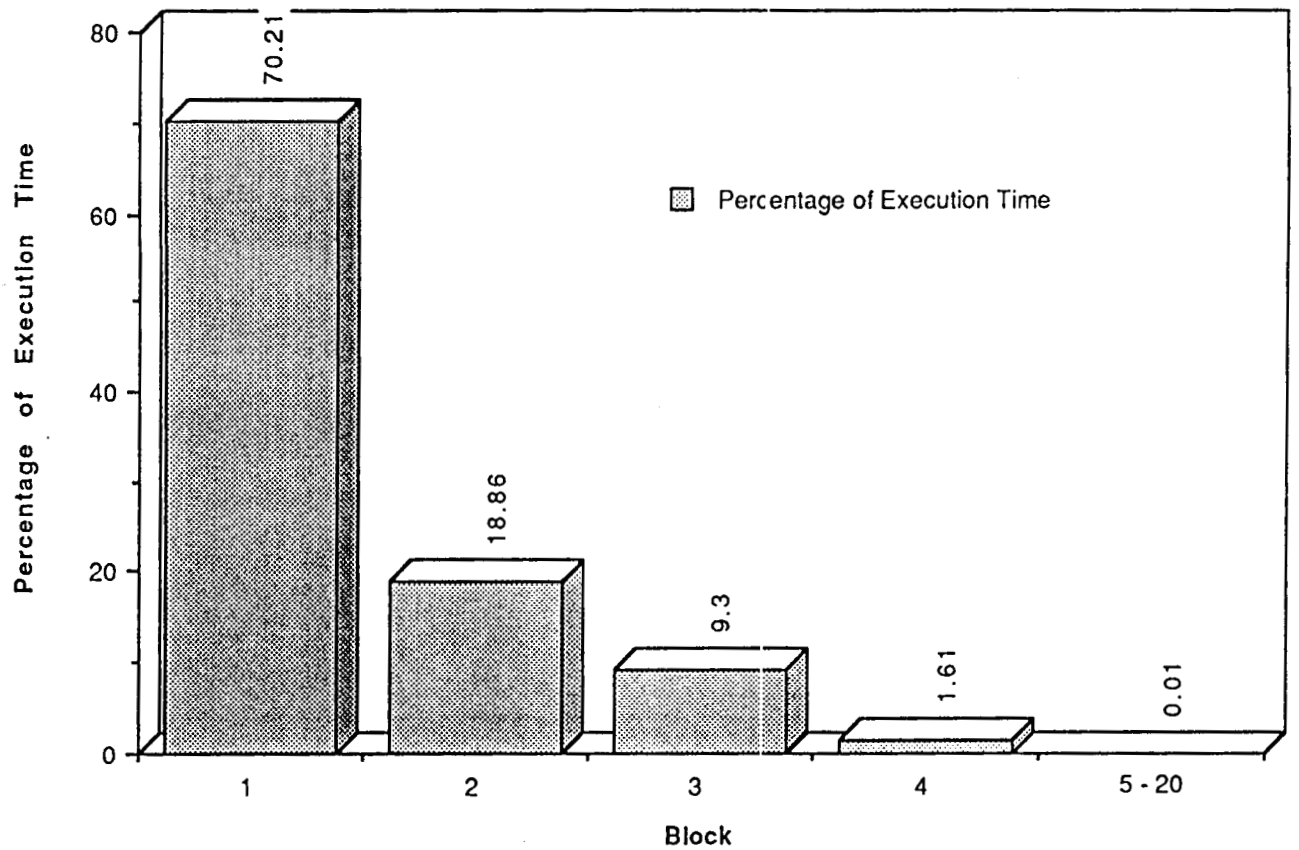


4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

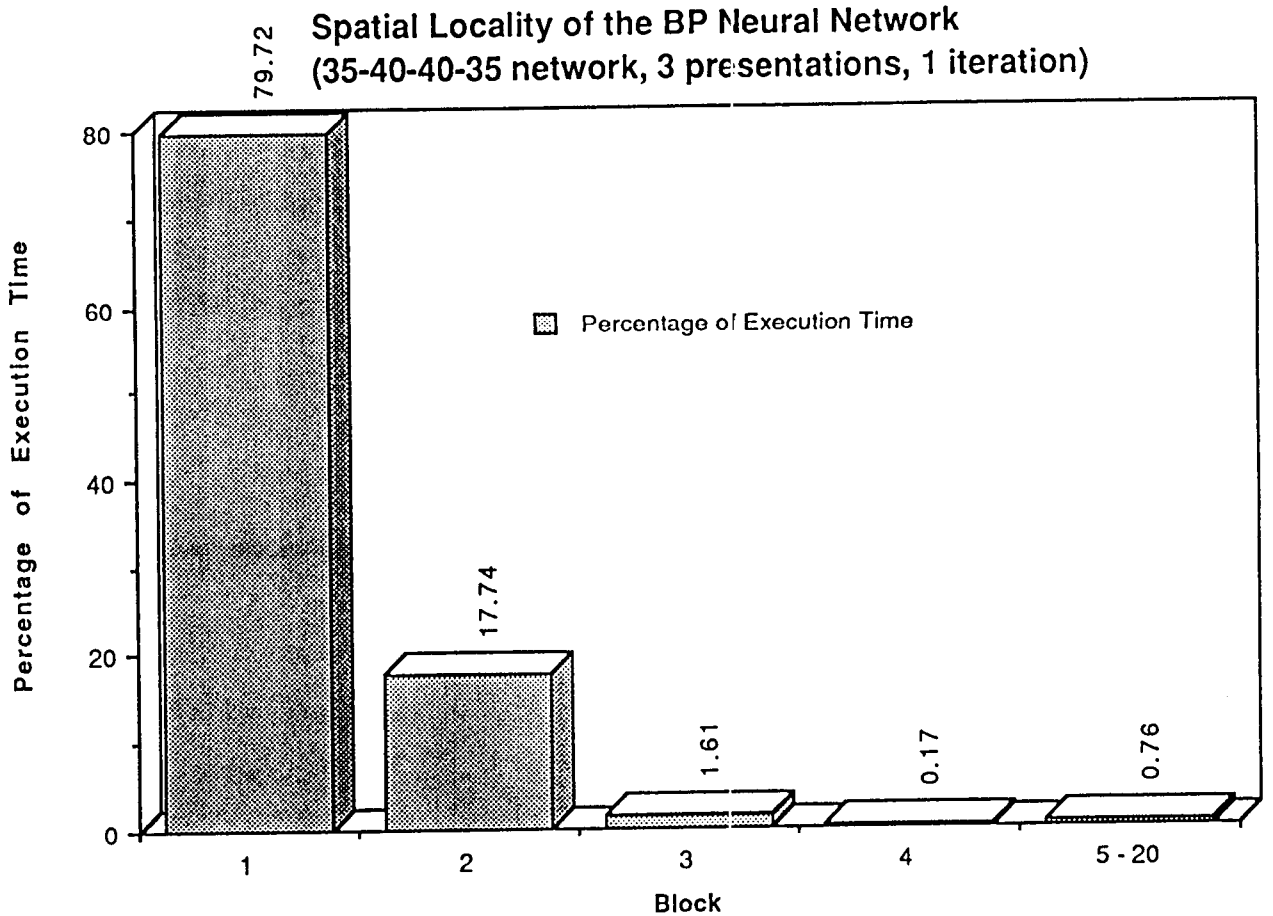
Spatial Locality of the BP Neural Network
(2-3-3-3 network, 30 presentations, 1000 iterations)



4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

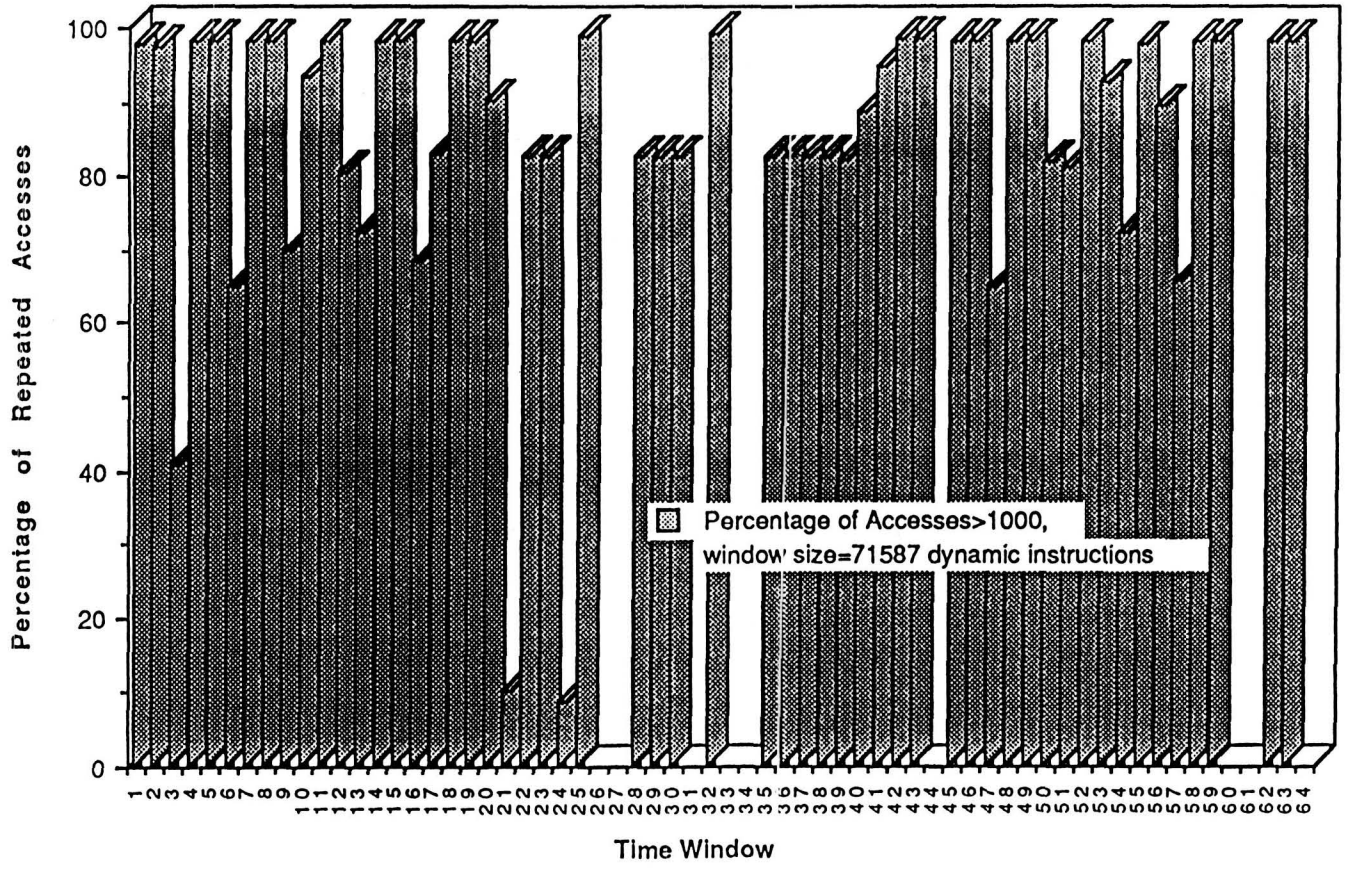


4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

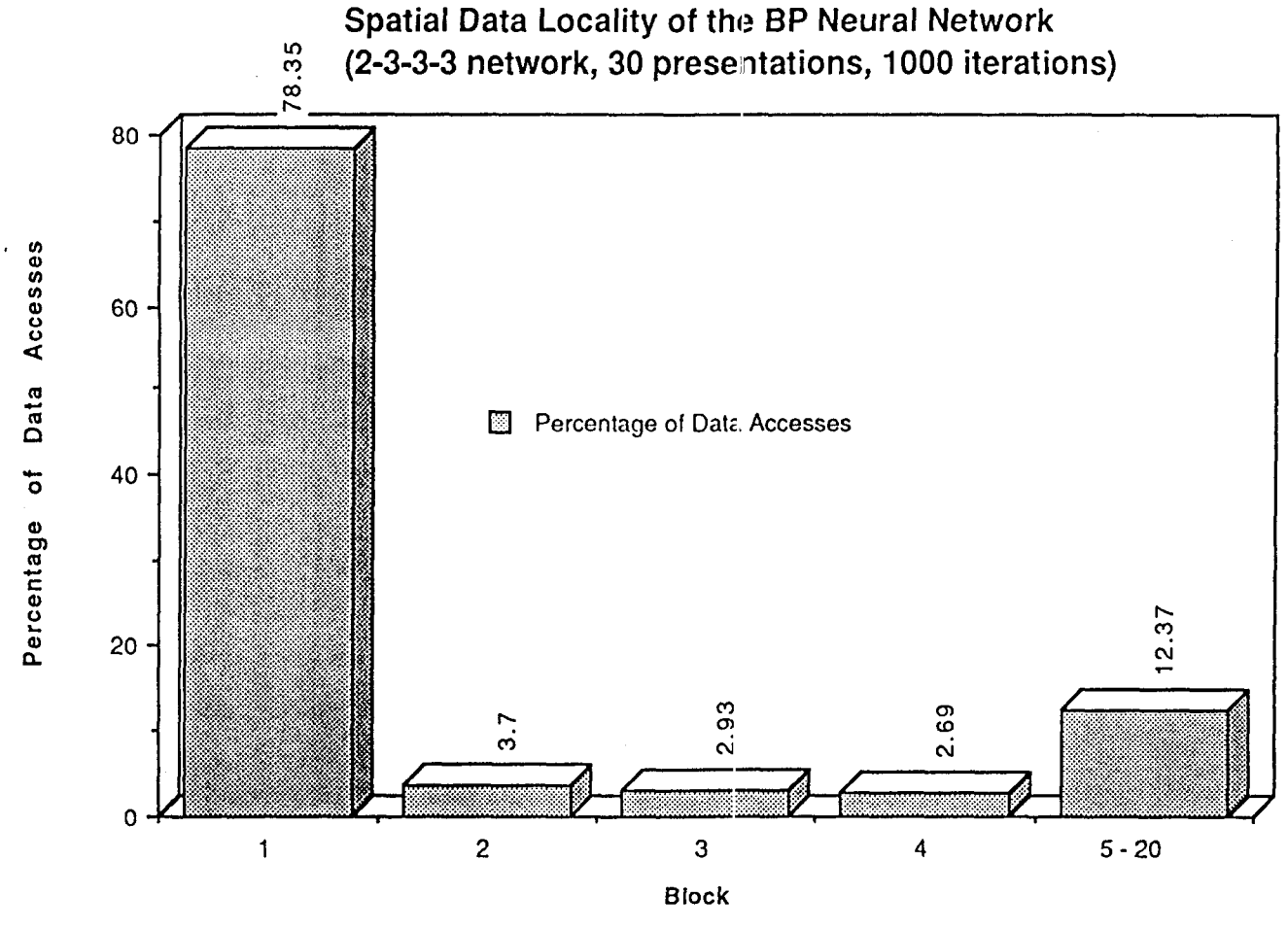
Temporal Locality of the BP Neural Network
(35-40-35-40 network, 3 presentations, 1 iteration)



4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

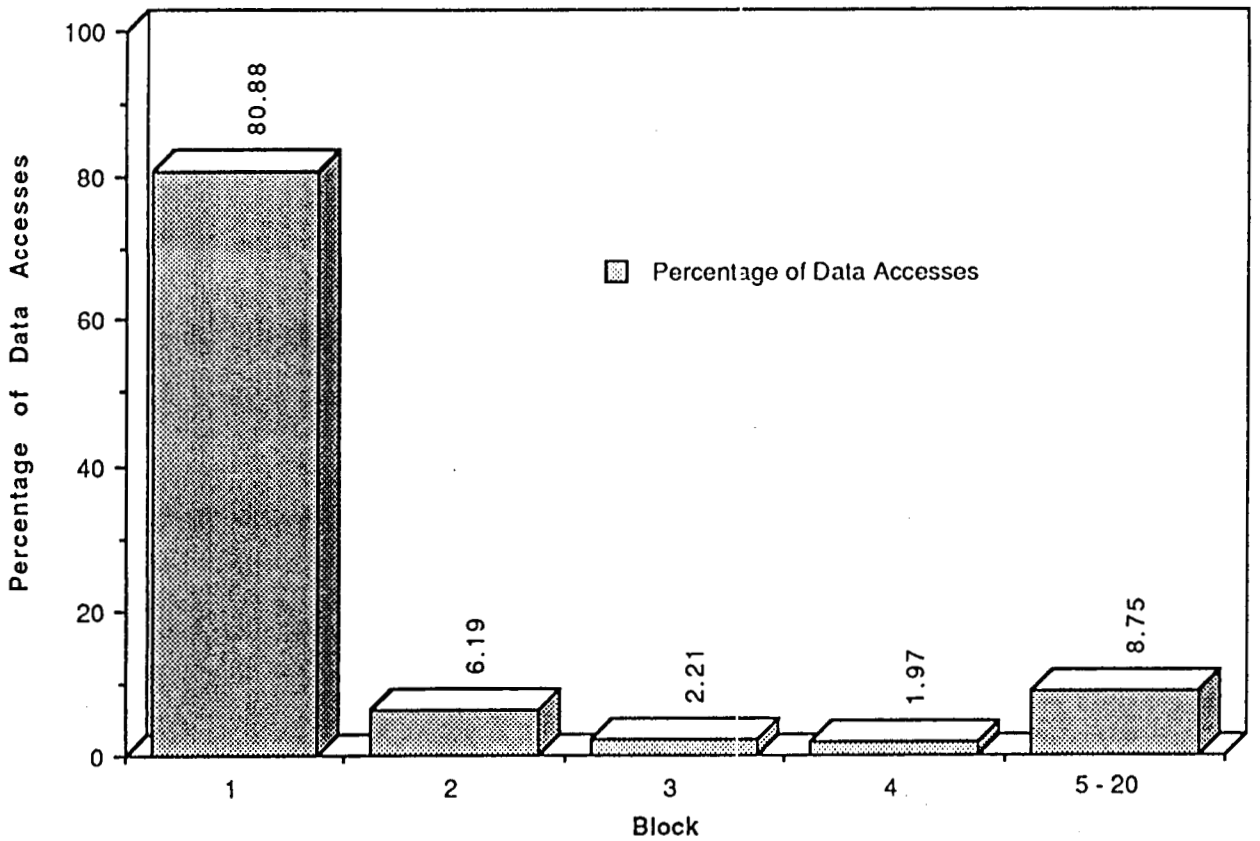


4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU

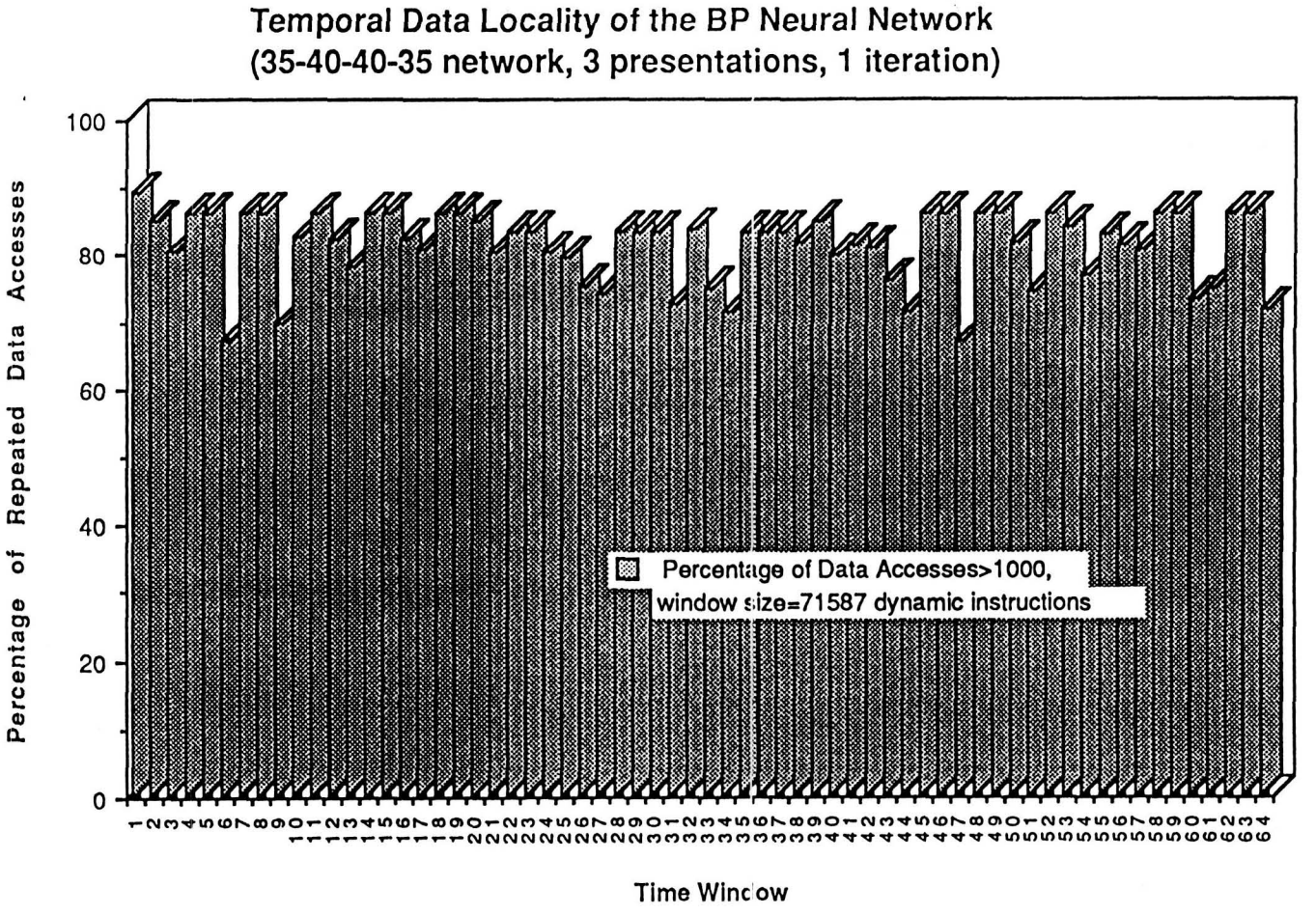
Spatial Data Locality of the BP Neural Network
(35-40-40-35 network, 3 presentations, 1 iteration)



4. Experimental Results: Backpropagation (Cont.)

CESDIS

GWU



5. Conclusions

- Temporal locality is, generally, higher for data.
- Locality properties are invariant to problem size.
- Back-propagation seems to have better spatial locality.
- In N-body, it seems that in the used algorithm, the critical path length grows more rapidly than parallelism with the increase in dynamic instructions count. This implies poor scalability for larger numbers of bodies.
- As the input increases in N-body, parallel instructions increase and start to dominate the execution producing less increase in variability (smoothability).
- In the Back-propagation, the independence in operating on the training sets resulted in close to linear increase in parallelism.
- The critical path length in Back-propagation, however, seems not to grow as rapidly with the increase in input. This resulted in decreasing smoothability.

Appendix-2

A sample output of SITA/Sitarun that shows the parallelism result for an instance of the developed Backpropagation code.

The output begins with some general information: the total number of operations encountered in the dynamic trace, the number of executed instructions, the number of useful instructions, and number of dynamic basic blocks. Then, for each machine model configured, a parallelism figure is given, along with a description of the model and some statistics on prediction rates and speculation depth. The parallelism is computed by dividing the total number of useful sequential instructions (the work) by the number of Parallel instructions needed to schedule these operations (the time).

```
PARALLELISM MEASUREMENT STATISTICS:

Seq. ops: 451086265 total, 450725414 exec., 413202312 useful; Total BB's: 29598872

M 1:     P = 803.636 (413202312/514166)  Ave. PI delay = 0.000
Ave. spec. depth = 0.000; Branch pred. = 0.000%; Jump pred. = 0.000%
        Rename registers = YES
        Unroll = NO
        Memory use:
                Stack:  DISAMBIGUATE    RENAME
                Heap:   DISAMBIGUATE    RENAME
                Resolution (in bytes) = 1
        Register sets = 7
        Non-strict stack usage = YES
        Control-dependence analysis = YES
        Procedure separation = YES
        Speculation depth = Infinite, conservative
        Branch prediction = Perfect
        Jump prediction = Perfect
        Processors = Infinite
        Pre-scheduling window = Infinite
        Post-scheduling queue = Infinite
        Latencies:      UNIFORM
Used 27185 1-byte cells in 311 pages - 87 cells/page (34.1% utilized)
```

Appendix-3

A sample output of SPAL/Spa-io that shows input/output system calls and related information.

Each input/output system call is printed on a new line. Each line consists of the number (order) of the system call, its type (read or write), its associated file, the amount of data read/written, the distance of the system call from the previous input/output system call in terms of the number of instructions, and the distance of the system call from the beginning of the trace in terms of the number of instructions. This information is useful in determining the place and intensity of input/output data transfers in the program.

** SysCall#	SysCallType	AssocFile	DataLength	Distance	Offset
5	w	fd1	69	135507	135507
6	w	fd1	69	14982	150489
7	w	fd1	69	14908	165397
8	w	fd1	69	14883	180280
9	w	fd1	69	14881	195161
10	w	fd1	69	14922	210083
11	w	fd1	69	14908	224991
12	w	fd1	69	14899	239890
13	w	fd1	69	14893	254783
14	w	fd1	69	14767	269550
15	w	fd1	69	14939	284489
16	w	fd1	69	14833	299322
17	w	fd1	69	14821	314143
18	w	fd1	69	14893	329036
19	w	fd1	69	14939	343975
20	w	fd1	69	14834	358809
21	w	fd1	69	14903	373712
22	w	fd1	69	14777	388489
23	w	fd1	69	14990	403479
24	w	fd1	69	14742	418221
25	w	fd1	69	14856	433077
26	w	fd1	69	14827	447904
27	w	fd1	69	15043	462947
28	w	fd1	69	14964	477911
29	w	fd1	69	14950	492861
30	w	fd1	69	14922	507783
31	w	fd1	69	14794	522577
32	w	fd1	69	14873	537450
33	w	fd1	69	14911	552361
34	w	fd1	69	14778	567139
35	w	fd1	69	14873	582012
36	w	fd1	69	14953	596965
37	w	fd1	69	14893	611858
38	w	fd1	69	14738	626596
39	w	fd1	69	14839	641435
40	w	fd1	69	14698	656133
41	w	fd1	69	14742	670875
42	w	fd1	69	14884	685759
43	w	fd1	69	14869	700628
44	w	fd1	69	14898	715526
45	w	fd1	1	223	715749
46	w	fd1	24	541921	1257670
47	w	fd1	34	8722	1266392
48	w	fd1	15	98054	1364446
49	w	fd1	25	9104	1373550
50	w	fd1	25	8993	1382543
51	w	fd1	25	8856	1391399
52	w	fd1	1	223	1391622
53	w	fd1	37	13473	1405095
54	w	fd1	37	13516	1418611
55	w	fd1	37	13327	1431938
56	w	fd1	1	223	1432161
57	w	fd1	37	13318	1445479
58	w	fd1	37	13565	1459044
59	w	fd1	37	13625	1472669
60	w	fd1	1	223	1472892
61	w	fd1	15	650	1473542
62	w	fd1	13	4349	1477891
63	w	fd1	13	4638	1482529
64	w	fd1	13	4637	1487166
65	w	fd1	1	223	1487389
66	w	fd1	13	4698	1492087
67	w	fd1	13	4482	1496569
68	w	fd1	13	4563	1501132
69	w	fd1	1	223	1501355

```
        70          w         fd1              13          4488      1505843
        71          w         fd1              13          4611      1510454
        72          w         fd1              13          4651      1515105
        73          w         fd1               1           223      1515328
        74          w         fd1              15           650      1515978
        75          w         fd1              25          5494      1521472
        76          w         fd1              35          9035      1530507
** Total system calls=80, read calls=0, write calls=72.
** Total data read=0, written=3344 characters.
```

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle

A Study on Evaluating Early High-Performance Computing Systems

5. Report Date

January 26, 1994

6. Performing Organization Code

7. Author(s)

Tarek El-Ghazawi, PI

8. Performing Organization Report No.

10. Work Unit No.

506-59-11

9. Performing Organization Name and Address

The George Washington University
Dept. of Electrical Engineering and Computer Science
Washington, DC 20052

11. Contract or Grant No.

NAS5-30428

13. Type of Report and Period Covered

Final, July 6 - December 31, 1993

12. Sponsoring Agency Name and Address

CESDIS (Center of Excellence in Space Data and Information Sciences)
NASA Goddard Space Flight Center
Code 930.5
Greenbelt, MD 20771

14. Sponsoring Agency Code

15. Supplementary Notes Task 31

CESDIS is operated by Universities Space Research Association (USRA), American City Building, Suite 212,10227 Wincopin Circle, Columbia, MD 21044

16. Abstract

Evaluating high-performance computing systems (HPCS) based on MPP architectures is a key part of the HPCC initiative. In order to provide HPCS designers with the insight needed to improve the next generation, one must correlate the performance measurements with the dynamic characteristics of the workload itself. This work develops an abstract model suitable for experimentally characterizing workloads based on a set of attributes that illustrate how the workloads stress the various architectural features. Meanwhile, the experimental characterization is conducted in a manner that isolates the effects of the machine loading or architecture on which the measurements are obtained, focusing only on the application. Characterization attributes are designed to reflect the applications' scalability, spatial and temporal locality of references for both instructions and data, and amount of work. Workload produced by NASA ESS Grand Challenge applications was experimentally characterized in order to explore the unique characteristics of this application domain.

17. Key Words (suggested by Author(s))	18. Distribution Statement

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price